# MIP - Proiect Laborator
# Clinică Veterinară

Tudose Alexandru

Ianuarie 2025

Modul în care voi aborda subiectele de barem nu va fi în ordinea dată, ci în ordinea aparițiilor și importanței acestora în implementarea mini-sistemului conceput.

## 1 Introducere în Java (Lab. 1 + 2)

Aplicația folosește conceptele fundamentale Java în mai multe moduri:

### 1.1 Tipuri de date și operații de bază

- Primitive: int pentru ID-uri, String pentru nume și condiții medicale
- Clase wrapper: Integer, String
- Operatori de comparare pentru verificarea ID-urilor

### 1.2 Structuri de control

- **if-else**: pentru validarea datelor de intrare și verificarea existenței proprietarilor/animalelor
- **while**: în meniul principal pentru rularea continuă a aplicației
- **switch**: pentru selectarea opțiunilor din meniu

### 1.3 Input/Output

- Scanner pentru citirea datelor de la tastatură
- System.out.println pentru afișarea meniurilor și mesajelor
- Gestionarea excepțiilor pentru input invalid

## 2 Interfețe (Lab. 6)

Proiectul folosește interfața IServices pentru a defini contractul serviciilor oferite de aplicație. Aceasta include operațiile fundamentale:

- addOwnerWithPet - adăugarea unui proprietar nou cu animalul său
- addPetToOwner - adăugarea unui animal nou la un proprietar existent
- healAnimal - vindecarea unui animal și eliminarea sa din sistem
- getRemainingOwners - obținerea listei proprietarilor rămași
- getRemainingAnimals - obținerea listei animalelor care mai necesită tratament

## 3 Clase Abstracte și Moșteniri (Lab. 4 + 5)

La baza ierarhiei de clase se află clasa abstractă Animal, care definește structura de bază pentru toate animalele din clinică:

- Atribute comune: id, name, species, medicalCondition
- Metodă abstractă makeSound() implementată specific de fiecare tip de animal
- Clasele Dog și Cat extind Animal, specificând comportamentul makeSound() și setând specia corespunzătoare

## 4 Colecții și Persistența Datelor (Lab. 3 + 8)

Aplicația utilizează diverse colecții Java pentru gestionarea datelor:

- List¡Animal¿ pentru lista de animale ale unui proprietar
- ArrayList pentru implementarea colecțiilor

Persistența datelor este realizată prin:

- Salvarea datelor în fișiere text
- Încărcarea automată a datelor la pornirea aplicației
- Actualizarea fișierelor la fiecare modificare a datelor

## 5 Testing (Lab. 7)

Am implementat teste unitare comprehensive pentru verificarea funcționalității:

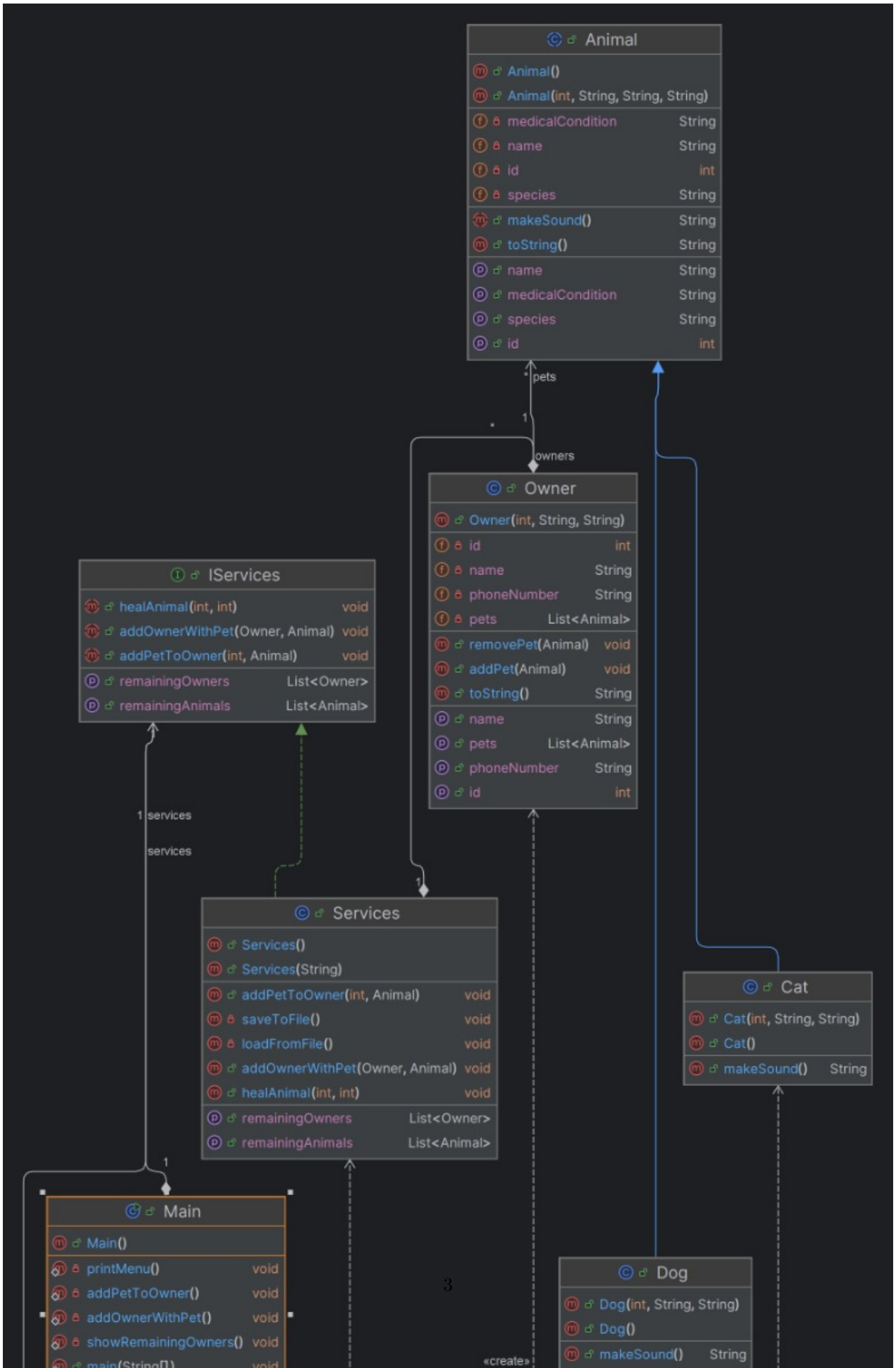
- Testarea adăugării proprietarilor și animalelor
- Verificarea procesului de vindecare
- Validarea persistenței datelor
- Testarea cazurilor de eroare

## 6 Diagrama UML (Lab. 9)

Diagrama UML atașată ilustrează structura completă a aplicației, evidențiind relațiile dintre clase și interfețe:

## 7 Concluzii

Proiectul implementează toate cerințele din barem, oferind o soluție funcțională pentru gestiunea unei clinici veterinare. Sistemul este modular, extensibil și ușor de întreținut datorită utilizării principiilor OOP și a design patterns-urilor adecvate.


Animal
Animal()
Animal(int, String, String, String)
medicalCondition String
name String
id int
species String
makeSound() String
toString() String
name String
medicalCondition String
species String
id int
pets
owners
Owner
Owner(int, String, String)
id int
name String
phoneNumber String
pets List<Animal>
removePet(Animal) void
addPet(Animal) void
toString() String
name String
pets List<Animal>
phoneNumber String
id int
IServices
healAnimal(int, int) void
addOwnerWithPet(Owner, Animal) void
addPetToOwner(int, Animal) void
remainingOwners List<Owner>
remainingAnimals List<Animal>
services
services
Services
Services()
Services(String)
addPetToOwner(int, Animal) void
saveToFile() void
loadFromFile() void
addOwnerWithPet(Owner, Animal) void
healAnimal(int, int) void
remainingOwners List<Owner>
remainingAnimals List<Animal>
Cat
Cat(int, String, String)
Cat()
makeSound() String
Main
Main()
printMenu() void
addPetToOwner() void
addOwnerWithPet() void
showRemainingOwners() void
Dog
Dog(int, String, String)
Dog()
makeSound() String
«create»